수치 해석학 과제5

1. About code

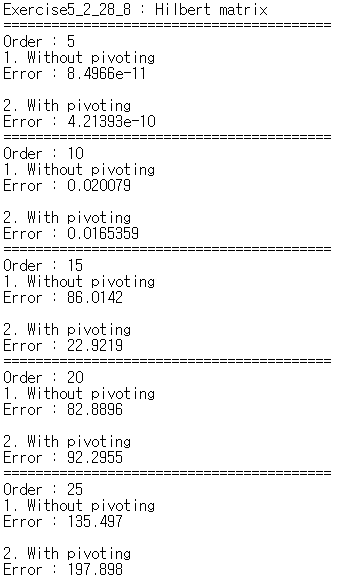
First, since I have to handle vectors and matrices in C++, I made a header file “Vector\_Matrix.h” which defines two classes FVector and FMatrix. In both classes, there are some useful member functions such as inner product, matrix multiplication etc.

Second, the class **LU\_Decompostion** is defined for LU decomposition and Gaussian elimination solver for linear equations. Since LU decomposition algorithm changes the input matrix when the algorithm overwrites the matrix, I made three LU functions. The function **void LU(FMatrix<double>& matrix)** just overwrites the input as the combination of L and U matrices. The function **void LU(const FMatrix<double>& matrix, FMatrix<double>& lu)** makes the input matrix lu into the LU decomposition of the input matrix. Finally, the function **void LU(const FMatrix<double>& matrix, FMatrix<double>& L, FMatrix<double>& U)** makes the input matrices L and U into the lower matrix and the upper matrix of the input matrix. Similarly, I made three LU\_pivot functions for using pivoting. For example, the function **LU\_pivot(FMatrix<double>& matrix, FVector<int>& permutation)** used the pivoting LU decomposition where the input vector ‘permutation’ is for row exchange. Finally, the function **void GE\_Solver(FMatrix<double>& matrix, FVector<double>&x, FVector<double>& b, bool pivot = false, bool copy = false)** is a solver for the linear equation : matrix\*x=b. If the input ‘pivot’ is true, the function uses LU decomposition with pivoting. If the input ‘copy’ is true, then the function does not change the input ‘matrix’ and uses a copy of the input for LU decomposition.

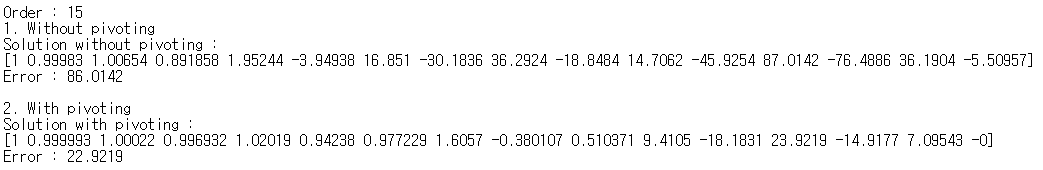
Finally, the class **Assignment5** is for solving Exercise 5.2.28.

1. Exercise 5.2.28 (8)

In this problem, I tested the least square approximation using Hilbert matrices with the order n=5, 10, 15, 20, 25. In the problem, the exact solution is a vector whose elements are all one. I compared -errors of LU decomposition without pivoting and with pivoting in each order. The result as follow:

When the order is 5, the solution is exactly same to the exact solution whether the pivoting is used or not. However, as the order increases, the errors of both cases extremely increases. Moreover, when the order is 20 or 25, the error with pivoting is more bigger than the error without pivoting. Since the Hilbert matrix becomes more ill-conditioned as the order increases, the errors of both cases increase as the order increases. Thus, the solutions obtained by LU decomposition are far away from the exact solution.

For example, the solutions with the order n=15 are given as follow:

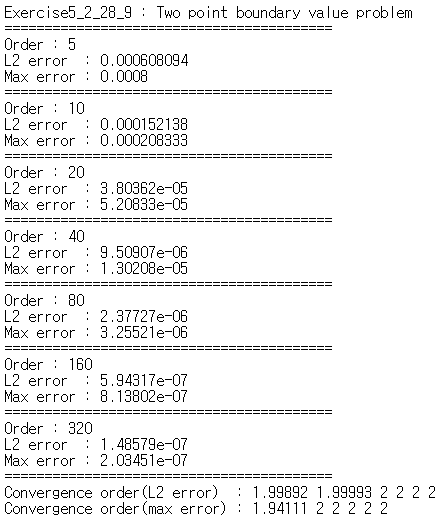


Some components of the solutions have extremely large absolute values, which determine the -errors. Moreover, the pivoting does not work when the order is big. When the pivoting is used, we fix the column and then compare entries to find the maximum absolute. Thus if the index of the column to be fixed is large, every entries along the column is much closed to zero. Thus the pivoting cannot handle the ill-condition.

1. Exercise 5.2.28 (9)

In this problem, I solved the two point boundary value problem using FDM scheme with LU decomposition. The function **void Dirichlet(double(\*f)(double x), double(\*u)(double x), int order, double& l2\_err, double& max\_err)** makes the inputs ‘l2\_err’ and ‘max\_err’ into -error and -error when the exact solution is the input function ‘u’ and the Laplace f is the input function ‘f’. Here, the -norm is calculated as where h is the mesh size.

I tested the code for the order (the number of mesh points) n=5, 10, 20, 40, 80, 160, 320. The result is as follow:



Then the convergence order for -error goes to 2 and the convergence order for -error goes to 2. Since the FDM scheme uses central difference to calculate derivatives, this is a second order scheme.